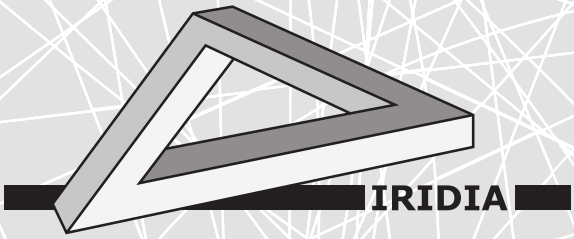


Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Tuning $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System with off-line and on-line methods

Paola Pellegrini, Thomas Stützle, and
Mauro Birattari

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2010-024

November 2010

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:
IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2010-024

Tuning $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System with off-line and on-line methods

Paola Pellegrini
Dipartimento di Matematica Applicata
Università Ca' Foscari Venezia, Venezia, Italia

Thomas Stützle and Mauro Birattari
IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
Contact: paolap@unive.it, stuetzle@ulb.ac.be, mbiro@ulb.ac.be

November 2010

Abstract

In this paper, we study two approaches for tuning parameters of optimization algorithms: *off-line* and *on-line* tuning. The goal of tuning is to find an appropriate configuration of an algorithm, where a configuration is a specific setting of all parameters. In off-line tuning, the appropriate configuration is chosen after a preliminary phase. In on-line tuning, the configuration might be changed during the solution process to adapt to the instance being solved and possibly to the state of the search. Parameter tuning is very important in swarm intelligence: the performance of many techniques is strongly influenced by the configuration chosen. In this paper, we study the performance of $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System when its parameters are tuned off-line and when they are tuned on-line. We consider one off-line and five on-line methods, and we tackle two combinatorial optimization problems. The results show that off-line tuning generally outperforms on-line tuning. In some cases, on-line tuning achieves better results than off-line tuning, but the best on-line method is not always the same.

1 Introduction

Many swarm intelligence techniques, such as ant colony optimization (Dorigo and Stützle, 2004) and particle swarm optimization (Clerc, 2006), have numerical and categorical parameters that have an impact on performance (Birattari, 2009). The appropriate configuration, i.e., the appropriate setting of all relevant parameters, depends on the problem instances to be tackled.

The approaches for finding appropriate configurations can be divided in two families: *off-line* and *on-line* tuning. In off-line tuning, the appropriate configuration is selected in a preliminary analysis. Traditionally, off-line tuning was performed manually, but recently a number of automatic procedures have been proposed (Birattari et al, 2002; Balaprakash et al, 2007; Adenso-Díaz and Laguna, 2006; Hutter et al, 2009). In on-line tuning, the configuration changes in dependence of the status of the search process while executing the algorithm. Also in this case, several automatic procedures have been proposed in the literature (Anghinolfi et al, 2008; Battiti et al, 2008; Lobo et al, 2007; Martens et al, 2007).

Off-line and on-line tuning have both merits and drawbacks. In particular, one of the main merits of off-line tuning is that it does not require knowledge on the specific algorithm to be tuned: it treats optimization algorithms as black-boxes. On the other hand, one of the main drawbacks

of off-line tuning is that some resources have to be devoted to tuning parameters before deploying the algorithm. In on-line tuning, no additional resource is necessary before actually tackling an instance, and a high flexibility is achieved by adapting the configuration to an instance, depending on the specific phase of the search. This flexibility is paid for in terms of design complexity: the on-line method must be incorporated in the implementation of the optimization algorithm. Thus, a very good understanding of all different features of the algorithm is necessary for obtaining an effective on-line method. Hence, an on-line method must be designed for a specific algorithm, and it cannot directly be exploited on different ones.

In this paper, we analyze the performance of off-line and on-line tuning under different experimental conditions, for testing whether the intuitions that typically drive the choice of one approach against the other may be supported by experimental evidence. In particular, we consider three a priori conjectures: a) off-line tuning outperforms on-line tuning for tackling homogeneous instances; b) the advantage of off-line tuning is mitigated when tackling heterogeneous instances, to the extent that it becomes a disadvantage when instances are highly heterogeneous; c) a large number of parameters to be tuned penalizes on-line tuning. We report the results of an extensive experimental analysis for refusing or corroborating these conjectures.

We consider $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) (Stützle and Hoos, 2000), one of the most successful ant colony optimization (ACO) algorithms, and we apply it to two well-studied combinatorial optimization problems: the traveling salesman problem (TSP) and the quadratic assignment problem (QAP). For each problem we consider several sets of instances with various levels of heterogeneity. We study the performance achieved by the algorithm when it is tuned off-line through the F-Race procedure (Birattari, 2003), and when it is tuned on-line. We test five on-line methods: one of them is based on a local search approach (Anghinolfi et al, 2008); the remaining four are based on a self-adaptive approach (Martens et al, 2007), which represents the state-of-the-art for on-line tuning in ACO. In self-adaptive approaches, the space of configurations is coupled with the solution space of the instance to be tackled, and the algorithm operates in the joint space so obtained. We analyze the performance of these on-line methods as a function of the number of parameters tuned.

In these experiments, off-line tuning achieves better performance than on-line tuning. Moreover, the results show that the relative performance of on-line methods depends both on the instances tackled and on the experimental conditions. This suggests that good results might be obtained by hybridizing off-line and on-line tuning: off-line tuning may help in selecting the most important parameters to be tuned on-line; then, on-line tuning may adapt the configuration to each specific instance and to the various phases of the search.

The rest of the paper is organized as follows. In Section 2, we describe the main characteristics of $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System and the specific algorithms we implement for solving the two problems tackled. In Section 3, we present the procedures we consider for off-line and on-line tuning. In Sections 4 and 5, we report the experimental setup and the results we obtain, respectively. Finally, in Section 6 we draw some conclusions.

2 $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System

In $\mathcal{MM}$AS, a colony of artificial ants explores the search space iteratively. Ants are independent agents that construct solutions incrementally, component by component. Ants communicate indirectly with each other through pheromone trails, biasing the search toward regions of the search space containing high quality solutions.

When applying an ACO algorithm, an optimization problem is typically mapped to a construction graph $G = (V, E)$, with V being the set of nodes and E being the set of edges connecting the nodes. Solution components may be represented either by nodes or by edges of this graph. In the following we describe the main procedures that characterize $\mathcal{MM}$AS, considering solution components associated to edges, and supposing that a minimization problem is to be solved. The description can be easily extended to the case of selecting components associated to nodes.

A pheromone trail τ_{ij} is associated to each edge $(i, j) \in E$: it represents the cumulated

knowledge of the colony on the convenience of including component (i, j) in solutions. At the end of each iteration, in which m ants construct one solution each, this cumulated knowledge is enriched by applying the pheromone update rule given in Equation (1). Some pheromone evaporates from each edge, and some is deposited on the edges belonging to the best solution, considering either the last iteration (iteration-best solution), the whole run (best-so-far solution), or the best since a reinitialization of the pheromone trails (restart-best solution) (Stützle and Hoos, 2000):

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \begin{cases} F(s_{best}), & \text{if edge } (i, j) \text{ is part of the best solution;} \\ 0, & \text{otherwise;} \end{cases} \tag{1}$$

where $F(s_{best})$ is the inverse of the cost of the best solution (recall that we are dealing with minimization problems here), and ρ is a parameter of the algorithm typically named evaporation rate ($0 < \rho < 1$). In addition, the pheromone strength is constantly maintained in the interval $[\tau_{min}, \tau_{max}]$. These bounds are functions of the state of the search. When pheromone trails are excessively concentrated they are re-initialized uniformly on all edges to favor exploration (Stützle and Hoos, 2000).

Exploiting the common knowledge represented by pheromone trails, ants construct solutions incrementally, by selecting at each step the edge to traverse. In $\mathcal{MM}$AS, this selection is done according to the random-proportional rule. Ant k, being in node i, moves to node $j \in N^k$ with probability

$$p_{ij} = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{h \in N^k} [\tau_{ih}]^\alpha \cdot [\eta_{ih}]^\beta}, \tag{2}$$

where α and β are parameters of the algorithm, η_{ij} is a heuristic measure representing the desirability of using edge (i, j) from a greedy point of view, and N^k is the set of nodes to which the ant can move to.

Typically, the m solutions generated by ants at each iteration are used as starting points for local search runs, one for each initial solution. The solutions returned are then used in the pheromone update as if they had been built by ants.

The procedures just described, characterizing $\mathcal{MM}$AS, may be used for designing algorithms for virtually any optimization problem. In Sections 2.1 and 2.2 we describe the specific implementation we use for the TSP and the QAP, respectively.

2.1 $\mathcal{MM}$AS for the TSP

The TSP consists in finding a minimum cost tour for visiting a set of cities exactly once, starting and ending at the same location. The cost of using each route connecting two cities is fixed. A TSP instance is mapped to a graph by associating a node to each city, and an edge with a predefined cost to each route. The objective of the problem is to construct a minimum cost Hamiltonian tour.

For solving the TSP, we use the algorithm implemented in the ACOTSP software (Stützle, 2002). The heuristic measure η_{ij} is the inverse of the cost of traversing the edge (i, j). The set of nodes in which ant k, being in node i, can move to (N^k) includes all nodes that belong to the candidate list associated to i and that have not been visited yet. The candidate list contains the nn nearest neighbors of node i, where nn is a parameter of the algorithm. The remaining nodes are included in N^k only if all nodes in the candidate list have already been visited.

The performance of $\mathcal{MM}$AS for the TSP can possibly be improved by using the pseudo-random proportional rule that is used by another ACO algorithm, namely ant colony system (ACS) (Dorigo and Gambardella, 1997). Thus, we exploit this rule here; with a probability q_0, the next node j to be inserted in an ant's partial solution is the most attractive node according to pheromone and heuristic measure:

$$j = \arg \max_{h \in N^k} \{\tau_{ih}^\alpha \eta_{ih}^\beta\}. \tag{3}$$

With probability $1 - q_0$, the ant uses the traditional random proportional rule of Equation (2). Thus, q_0 is a further parameter of the algorithm, with $0 \leq q_0 < 1$.

The 2-opt local search is applied to all solutions constructed by ants.

2.2 $\mathcal{MM}$AS for the QAP

The QAP consists in finding a minimum cost assignment of a set of facilities to a set of locations. A fixed distance is given for each pair of locations, and a fixed flow is given for each pair of facilities. The cost contribution for a pair of assignments of two facilities to two different locations is given by the product between the flow exchanged by the facilities and the distance of their respective locations. The total cost in the QAP is the sum of all cost contributions by all pairs of assignments. A QAP instance is mapped to a graph by associating nodes to facilities and locations: using edge (i, j) in the solution construction corresponds to assigning facility i to location j.

The implementation of $\mathcal{MM}$AS for the QAP used in this paper is described by Stützle and Hoos (2000). No heuristic measure is associated to edges; thus, parameter β is not considered here.

The algorithm strictly follows the general framework of $\mathcal{MM}$AS described at the beginning of the current section. Depending on the characteristics of the set of instances tackled, the best performing local search may vary (Stützle and Hoos, 2000). Thus, local search is a further parameter of the algorithm. In the following it will be referred to as ls.

3 Off-line and on-line tuning: the methods considered

We study the results achieved by $\mathcal{MM}$AS when its parameters are tuned off-line and when they are tuned on-line. In the following, let P be the vector of all H parameters of the algorithm, and P_t be the vector of h parameters to be tuned. The parameters in P_t are a subset of those in P. Tuning methods operate in a configuration space determined *a priori*. Each possible setting of a parameter p is given by a vector S_p. If p is a numerical parameter, a setting is a number. We assume that the possible values are obtained by a discretization of the feasible intervals in the case of real-valued parameters, or by a selection of representative values in the case of integer parameters. The possible values are listed in S_p in increasing order. If p is a categorical parameter we use an arbitrary order. The configuration space is then given as the set of all possible combinations of parameter settings.

For what concerns off-line tuning, F-Race (Birattari, 2009; Birattari et al, 2002) appears to be a convenient choice for tuning the relatively small number of candidate configurations considered. Before solving the instances of interest, F-Race exploits a set of tuning instances with characteristics similar to those of the instances to be tackled. One step of F-Race consists in evaluating each configuration on one tuning instance. After each step, F-Race discards any configuration that is worse than at least another one.

For what concerns on-line tuning, we consider five different methods. The first one, LS, is based on the exploration of the configuration space using a naive local search approach (Anghinolfi et al, 2008). LS evaluates at each step one reference configuration and the neighbors of it. The neighborhood includes all configurations that differ in the setting of exactly one parameter p from the reference one. If the setting of this parameter in the reference configuration is the i^{th} setting in the vector of possible ones S_p, the neighbors are the ones taking the $(i-1)^{\text{st}}$ and the $(i+1)^{\text{st}}$ settings. If either the $(i-1)^{\text{st}}$ or the $(i+1)^{\text{st}}$ setting do not exist in S_p, which means that the reference configuration includes either the first or the last setting in S_p, one neighbor is not valid, and LS doubles the reference configuration. The reference configuration is replaced by one of the neighbors if it is not the best performing. LS evaluates the configurations by splitting the ant colony in as many groups as is the size of the neighborhood, and by having each group of ants building solutions using a different configuration. The performance of each configuration is defined as the value of the best solution found by the corresponding group of ants. LS evaluates each candidate configuration for 10 iterations of $\mathcal{MM}$AS before deciding whether changing the

reference configuration or not, as done by Anghinolfi et al (2008). In case of ties, they are resolved randomly

The remaining four on-line methods are self-adaptive approaches. In self-adaptive approaches the mechanism that is used to modify configurations is the same as that underlying the algorithm to be tuned (Eiben et al, 2007). This can be realized by associating to each parameter p_i to be tuned a set of nodes V'_{p_i} in the construction graph, corresponding to all possible settings in S_{p_i}. This results in an additional set of nodes

$$V' = \bigcup_{p_i \in P_t} V'_{p_i}. \tag{4}$$

Each node associated to a parameter is connected through an edge to each node associated to the following parameter in P_t resulting in edge set E'. Each node associated to the last parameter is connected to all nodes in the original set V. Pheromone trails are associated to all edges in $E \cup E'$ and they all are managed with the update rule reported in Equation (1). The configuration to be used is built by the search mechanism that is used in the ACO algorithm, by selecting one node in each set V'_{p_i}. Multiple attempts have been made for exploiting efficiently the self-adaptive approach in ACO algorithms: Martens et al (2007); Randall (2004); Förster et al (2007) and Khichane et al (2009) have proposed different interpretations of this approach. They differ in two main points. On one hand, parameters have been considered either independently or interdependently from one another. If parameters are considered independently, pheromone trails are associated to the nodes in V'. In this way, a setting for one parameter is chosen independently of the other settings. If pheromone is on the edges, interactions among parameters may be taken into account. In all the on-line methods used in this paper, parameters are considered as interdependent. On the other hand, parameters have been managed either at the colony or at the ant level. Three methods we implemented for this study manage parameters at the colony level: at each iteration the same setting is used for the whole colony. The three methods differ in the ratio used for choosing the edges of E' on which pheromone is deposited. The first method, SAc, reinforces pheromone on the configuration with which the best-so-far, the iteration-best, or the restart-best solution was found, according to the schedule defined by Dorigo and Stützle (2004). The second and the third methods, SAcb and SAcm, reinforce pheromone on the best configuration among those used in the last 25 iterations. SAcb uses the configuration with which the algorithm found the best solution; SAcm the configuration with which the algorithm found the set of solutions with the best mean cost. The fourth method, SAa, manages the parameters at the ant level, that is, at each iteration ants may use different configurations.

Both, ρ and m are colony-wise parameters. Thus they cannot be tuned on-line by the methods that use multiple configurations in each iteration of $\mathcal{MM}$AS, that is, methods where parameters are chosen based on the ant-level. Hence SAc, SAcb and SAcm may tune all the parameters described in Section 2: $P = (\alpha, \beta, \rho, m, q_0, n)$ for the TSP, and $P = (\alpha, \rho, m)$ for the QAP. SAa and LS, instead, may tune fewer parameters: $P = (\alpha, \beta, q_0, n)$ for the TSP, and $P = (\alpha)$ for the QAP.

4 Experimental Setup

In the experimental analysis, we consider multiple versions of $\mathcal{MM}$AS that differ in the way parameters are managed and in the initial configuration used:

1. the configuration is maintained fixed throughout the instances and throughout the runs. In this case, we distinguish two configurations :

   - *literature* (L): the configuration used is the one suggested in the literature (Stützle and Hoos, 2000; Dorigo and Stützle, 2004);
   - *off-line* (OFF): the configuration used is the one selected by F-Race.

Table 1: Values that can be chosen for each parameter for the TSP and the QAP. The values reported in bold type are the ones suggested in the literature (Stützle and Hoos, 2000; Dorigo and Stützle, 2004). They are used in the literature and on-line versions.

TSP $P = (q_0, \beta, \rho, m, \alpha, nn)$	
parameter	settings (S)
q_0	**0.0**, 0.25, 0.5, 0.75, 0.9
β	1, **2**, 3, 5, 10
ρ	0.1, **0.2**, 0.3, 0.5, 0.7 6
m	5, 10, **25**, 50, 100
α	0.5, **1**, 1.5, 2, 3
nn	10, **20**, 40, 60, 80
QAP $P = (m, \rho, \alpha)$	
parameter	settings (S)
m	1,2,**5**,8,16
ρ	**0.2**,0.4,0.6,0.8
α	0.5, **1**, 1.5, 2, 3
l	0=first improvement (don't look bits), 1=first improvement (no don't look bits), **2=best improvement**, **3=tabu search runs of length $2n$**, 4=tabu search runs of length $6n$

2. the configuration is tuned on-line according to one of the methods described in Section 3: ON $\in$ {LS, SAa, SAc, SAcb, SAcm}:

   - *literature + on-line* (L+ON): the initial configuration is the one reported in the literature (Stützle and Hoos, 2000; Dorigo and Stützle, 2004);
   - *off-line + on-line* (OFF+ON): the initial configuration is the one selected by F-Race.

We evaluate on-line methods as a function of the number of parameters tuned h, for refusing or corroborating the conjecture stating that the performance of these methods worsens if several parameters are to be tuned. For each number of parameters tuned, we test all vectors of cardinality h (recall that the cardinality of P is indicated as H). The total number of combinations, and thus the total number of P_t's for each h is: $\begin{pmatrix} H \\ h \end{pmatrix}$. In the discussion of the results we report the number of parameters tuned immediately after the acronym of the on-line method, between parenthesis, e.g, L-SAc(3) means that L-SAc has to tune three parameters.

In the experiments, we consider the same sets of parameter settings for all tuning methods to avoid a bias in favor of one method. In fact, different configuration spaces may influence the relative performance of the methods. In Table 1, we report the vector S of possible values of each parameters. The configuration space includes all combinations of the different settings; it amounts to 15,625 candidates for the TSP, and 500 for the QAP. The configurations suggested in the literature are reported in bold type. We show the vector of parameters of the algorithm next to the problem acronym. When P_t does not include all parameters of P, the same order is maintained, after deleting the parameters not included in the specific subset tested.

For the QAP, the literature suggests to use 2-opt with best improvement for structured instances, and tabu search with run length $2n$ for unstructured instances (Stützle and Hoos, 2000). Not being possible to identify a generally appropriate local search, we add parameter ls to the set P considered for off-line tuning. In on-line tuning, instead, P never includes ls: preliminary results show that the on-line methods implemented are penalized by the presence of ls in the set of parameters to be tuned. The difficulty for tuning on-line local search is that both the computational time and the solution quality may differ for each settings. So, there is a trade-off between computational time and solution quality. Adjusting on-line methods to take this trade-off into account would require some additional effort.

We compare off-line and on-line tuning on multiple sets of instances for each problem, performing both short and long runs. The results reported for each version of $\mathcal{MM}$AS are obtained with one single run on 100 instances for each set and for each run length (Birattari, 2004; Birattari and Dorigo, 2007). We perform the experiments on Xeon E5410 quad core 2.33GHz processors with

Table 2: Sets of instances considered for the TSP. $U(a, b)$ indicates that a number was randomly drawn between a and b for each instance, according to a uniform probability distribution. The second part reports the configuration selected by F-Race for short and long runs.

set	number of nodes	spatial distribution	F-Race selection	α	β	ρ	q_0	m	nn
$TSP(2000, u)$	2000	uniform	short	1	5	0.75	0.5	25	20
			long	1	3	0.5	0.5	100	20
$TSP(2000, c)$	2000	clustered	short	2	1	0.25	0.75	25	40
$TSP(2000, x)$	2000	uniform and clustered	short	1	1	0.25	0.9	25	20
$TSP(x, u)$	$U(1000, 2000)$	uniform	short	1	5	0.75	0.25	50	20
$TSP(x, c)$	$U(1000, 2000)$	clustered	short	2	2	0.25	0.75	50	40
$TSP(x, x)$	$U(1000, 2000)$	uniform and clustered	short	1	1	0.25	0.9	50	20

2x6 MB L2-Cache and 8 GB RAM, running under the Linux Rocks Cluster Distribution, after compiling the code with `gcc`, version 3.4.

For each run length, we execute F-Race on 1000 tuning instances of each set. These instances do not include those used in the evaluations of the tuning methods. Each run of F-Race terminates when either all tuning instances are solved, or a fixed number of runs of $\mathcal{MM}$AS are executed (156,250 in the TSP and 7,500 in the QAP).

To examine the impact of the heterogeneity of instance sets on tuning methods, for the TSP we define six sets of instances with different number of cities and different spatial distribution of the cities. All instances are generated through portgen, the instance generator used in the 8th DIMACS Challenge on the TSP (Johnson et al, 2001). The characteristics of each set are described in Table 2. In second part of this Table we report the configuration selected by F-Race for short and long runs. Hereafter, we will refer to a set of TSP instances as *TSP* followed by a parenthesis indicating the number of cities included and their spatial distribution. When either the number of cities or their spatial distribution are not the same in all instances, we report an x in the corresponding position. For example, if a set includes instances with 2000 cities that can be either uniformly distributed in the space or grouped in clusters, we use the acronym $TSP(2000, x)$. The different sets have different levels of heterogeneity:

- we call a set homogeneous if all instances have the same number of cities and the same spatial distribution, as in $TSP(2000, u)$ and $TSP(2000, c)$;
- we call a set heterogeneous if neither the number of cities nor their spatial distribution is the same in all instances, as in set $TSP(x, x)$;
- at an intermediate level between those two extremes, we define instance sets where either the number of cities or their spatial distribution is not the same in all instances, as in sets $TSP(2000, x)$, $TSP(x, u)$ and $TSP(x, c)$. For convenience we refer to these sets as semi-heterogeneous.

The run length is 10 and 60 CPU seconds for short and long runs, respectively. In short runs, the literature version completes between 100 and 120 iterations on instances of set $TSP(2000, u)$. Long runs last 60 CPU seconds. We performed long runs only on instances of set $TSP(2000, u)$.

For the QAP, we consider 28 sets of instances obtained through the instance generator described by Stützle and Fernandes (2004). We generate instances of three sizes: 60, 80 and 100. For each size, we generate both unstructured (RR) and structured (ES) instances. In unstructured instances, the entries of both distance and flow matrices are random numbers uniformly distributed in the interval $[0, 99]$ (Taillard, 1991). In structured instances, the entries of the distance matrix are the Euclidean distances of points whose coordinates are drawn in a square 100×100 according to a uniform distribution. The entries are rounded to the nearest integer. For what concerns the flow matrix, first a set of points are randomly located in a square of size 100×100. For each pair of points, if the distance is longer than a predefined threshold t, then the flow is set to zero;

otherwise it is equal to a value x resulting from the following procedure: i) consider a random number $x_1 \in [0, 0.7]$; ii) compute $x_2 = -\ln x_1$; iv) set $x = 100x_2^{2.5}$. The result of this procedure is a matrix in which the number of entries for each value is decreasing as a function of the value itself. We generate instances by setting different values of parameter t. The peculiarities of each set are described in Table 3. In the second half of Table 3 we report the configuration selected by F-Race for short and long runs. Hereafter we will refer to the sets of QAP instances as QAP followed by a parenthesis indicating the size of the instances and the characteristics of the matrices. As in the TSP, when the instances in a set have different characteristics, an x is reported in the corresponding position. For example, the set of instances generated inserting random entries in the distance and flow matrices of size 60, 80 or 100, is indicated as $QAP(x, RR)$. Also for the QAP we define a scale of heterogeneity for the sets of instances:

- an instance set is homogeneous if all instances have the same size and characteristics of the distance and the flow matrices, as in the first 18 sets in Table 3;
- an instance set is heterogeneous if neither the size nor characteristics of the distance and the flow matrices are the same in all instances, as in last set in Table 3;
- semi-heterogeneous refers to instance sets where either the size or instance characteristics, either RR or ES, are not the same in all instances, as in the 19^{th} to the 27^{th} set in Table 3.

The run length is 17 and 29 CPU seconds for short and long runs, respectively. For determining these values we run the literature version on instances of size 60, 80 and 100, considering as stopping criterion the number of iterations performed. Short runs are stopped after 200 iterations; long runs after 600. In the experiments, the time available for solving one instance is the average computational time used in these runs.

5 Experimental Results

We compare off-line and on-line methods when all parameters of $\mathcal{MM}$AS are tuned off-line, and either one or more parameters are tuned on-line. As said before, beyond the relative performance of off-line and on-line tuning, we also study the impact of the number of parameters to be tuned on the performance of on-line tuning methods. Therefore, for each number of parameters to be tuned, w e consider two possible contexts:

1. **no *a priori* knowledge** on the best parameter(s) to be tuned. This context simulates the situation in which one has no information on the parameter(s) that can be most profitably tuned on-line. The expected performance in this case is neither the best achievable nor the worst. For mimicking this situation we consider the average result over all combinations of parameters that can be tuned;
2. **perfect *a posteriori* knowledge** on the best parameter(s) to be tuned. This context simulates the situation in which one perfectly knows which are the most important parameters to be tuned on-line for achieving the best performance. For mimicking this situation we consider only the combination of parameters that results to be the best *a posteriori*.

For each problem, we report the results of all tuning method for two sets of instances. In Pellegrini et al (2010a), we depict the results for all the sets described in section 4.

5.1 Traveling Salesman Problem

We report the results obtained in short runs by the off-line method and by the five on-line methods on the instances of sets $TSP(2000, u)$ and $TSP(x, x)$. They are represented in Figures 1 and 3 for the two contexts considered: no *a priori* and perfect *a posteriori* knowledge on the best vector to tune. We present the results in terms of position in the ranking achieved by the different variants of the algorithm within the instances of each set. We test the significance of the differences with

Table 3: Sets of instances considered for the QAP. The second part reports the configuration selected by F-Race for short and long runs.

set	size	type	F-Race selection				
			α	ρ	l	m	
$QAP(60, RR)$	60	unstructured	short	1	0.8	4	5
			long	1	0.4	4	2
$QAP(60, ES10)$	60	structured, $t = 10$	short	1	0.4	2	8
			long	1	0.4	2	16
$QAP(60, ES20)$	60	structured, $t = 20$	short	1	0.4	2	5
			long	1	0.4	2	8
$QAP(60, ES42)$	60	structured, $t = 42$	short	1.5	0.4	0	8
			long	1	0.4	2	2
$QAP(60, ES72)$	60	structured, $t = 72$	short	1.5	0.2	0	2
			long	1.5	0.2	0	5
$QAP(60, ES100)$	60	structured, $t = 100$	short	1	0.6	0	8
			long	1.5	0.4	0	2
$QAP(80, RR)$	80	unstructured	short	1	0.6	3	2
			long	1	0.4	3	1
$QAP(80, ES10)$	80	structured, $t = 10$	short	1	0.4	2	5
			long	1	0.4	2	8
$QAP(80, ES20)$	80	structured, $t = 20$	short	1	0.4	2	2
			long	1	0.4	2	2
$QAP(80, ES42)$	80	structured, $t = 42$	short	1	0.4	2	2
			long	1	0.4	2	5
$QAP(80, ES72)$	80	structured, $t = 72$	short	1.5	0.4	0	5
			long	1.5	0.2	0	5
$QAP(80, ES100)$	80	structured, $t = 100$	short	1.5	0.4	0	2
			long	1.5	0.2	0	2
$QAP(100, RR)$	100	unstructured	short	1	0.8	3	2
			long	1	0.6	3	1
$QAP(100, ES10)$	100	structured, $t = 10$	short	1	0.2	2	5
			long	1	0.4	2	5
$QAP(100, ES20)$	100	structured, $t = 20$	short	1	0.4	0	5
			long	1	0.4	0	5
$QAP(100, ES42)$	100	structured, $t = 42$	short	1	0.4	2	2
			long	1	0.4	2	2
$QAP(100, ES72)$	100	structured, $t = 72$	short	1.5	0.4	0	5
			long	1.5	0.4	0	5
$QAP(100, ES100)$	100	structured, $t = 100$	short	1.5	0.4	0	2
			long	1.5	0.4	0	2
$QAP(60, Ex)$	60	structured, t	short	1	0.4	2	5
		$\in \{10, 20, 42, 72, 100\}$	long	1	0.4	2	8
$QAP(80, Ex)$	80	structured, t	short	1	0.4	2	2
		$\in \{10, 20, 42, 72, 100\}$	long	1	0.4	2	5
$QAP(80, xx)$	80	unstructured or	short	1	0.4	3	2
		structured, $t = 10$	long	1	0.4	3	2
$QAP(100, Ex)$	100	structured, t	short	1	0.4	2	2
		$\in \{10, 20, 42, 72, 100\}$	long	1	0.4	2	2
$QAP(x, ES10)$	$\in \{60, 80, 100\}$	structured, $t = 10$	short	1	0.4	2	8
			long	1	0.4	2	5
$QAP(x, ES20)$	$\in \{60, 80, 100\}$	structured, $t = 20$	short	1.5	0.2	0	5
			long	1	0.2	2	5
$QAP(x, ES42)$	$\in \{60, 80, 100\}$	structured, $t = 42$	short	1.5	0.4	0	8
			long	1.5	0.4	0	5
$QAP(x, ES72)$	$\in \{60, 80, 100\}$	structured, $t = 72$	short	1.5	0.2	0	2
			long	1.5	0.4	0	5
$QAP(x, ES100)$	$\in \{60, 80, 100\}$	structured, $t = 100$	short	1.5	0.2	0	5
			long	1.5	0.4	0	5
$QAP(x, Ex)$	$\in \{60, 80, 100\}$	structured, t	short	1	0.4	2	5
		$\in \{10, 20, 42, 72, 100\}$	long	1	0.4	2	5

the Friedman test for all-pairwise comparisons, and in the plots we report the 95% simultaneous confidence intervals: for each variant we depict the median rank over all instances, and we represent the bounds of the corresponding confidence interval; if the intervals of two variants overlap, then the difference among the variants is not statistically significant. The order in which the acronyms are listed reflects their median ranking (Chiarandini, 2005). Moreover, each on-line method is evaluated as a function of the number of parameters tuned. We present the results used in this analysis for the same two sets of instances, in Figures 2 and 4, in which we focus on the performance of one well performing on-line method, namely SAc. We depict the boxplot of the rankings obtained in the 100 instances of each set. In Pellegrini et al (2010b), we reported the results obtained with the same analysis on smaller sets of instances.

5.1.1 Experiment 1: no *a priori* knowledge on the best parameter(s) to be tuned.

When we consider the context in which the implementer has no *a priori* knowledge on the impact of the different parameters on the behavior of on-line tuning methods, we achieve the results represented in Figure 1. The off-line version performs significantly better than all on-line versions on both sets, $TSP(2000, u)$ and $TSP(x, x)$. The relative performance of the approaches is the same irrespective of the level of heterogeneity of the sets of instances.

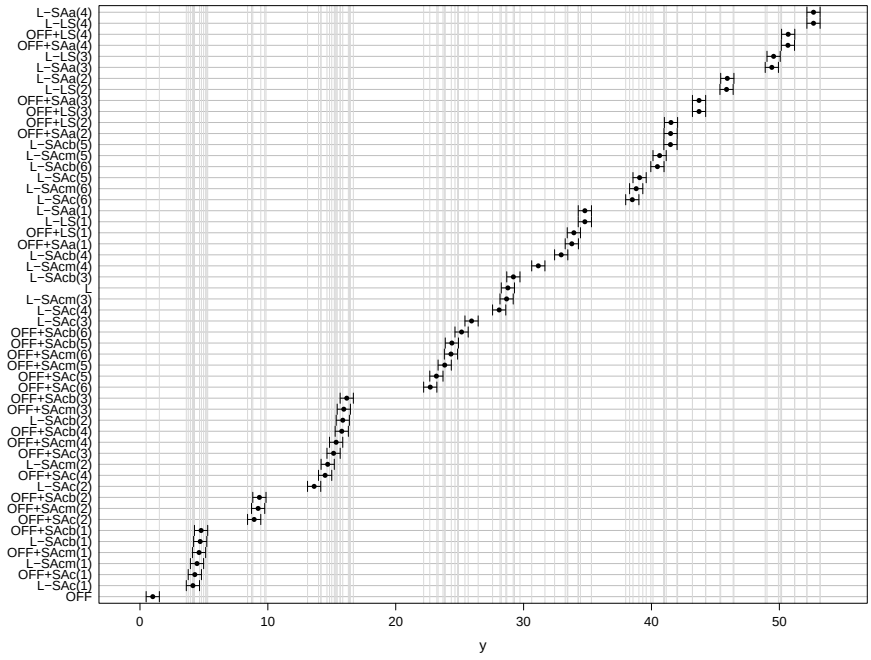
Focusing on the relative performance of on-line versions, using the configuration selected by F-Race as the initial one appears quite advantageous: the average ranking of off-line + on-line versions is 22.40 and 22.06 in sets $TSP(2000, u)$ and $TSP(x, x)$, respectively, while the average ranking of the literature + on-line ones is 30.61 and 30.94. This trends can be detected also by observing the lowest part of both graphics in Figure 1, even if the best performing on-line method on set $TSP(2000, u)$ is an on-line version. Self-adaptive approaches with parameters managed at the colony level outperform quite constantly the other on-line methods. The methods with parameters tuned at the ant level, LS and SAa, are always in the last positions of the ranking, regardless the number of parameters tuned. The average ranking on set $TSP(2000, u)$ is 19.57 for the methods that manage parameters at the colony level, and 42.08 for those that manage parameters at the ant level. The average rankings in set $TSP(x, x)$ are 19.44 and 42.38, respectively.

The results achieved in the sets of instances not reported here suggest the same observations: the off-line version is always the best performing, and the results achieved by off-line + on-line versions are in average better than those of the literature + on-line ones. The best results are found by OFF+SAcm on set $TSP(x, c)$ and by OFF+SAc in the remaining three sets. The results for long runs on set $TSP(2000, u)$ are qualitatively equivalent, a part from two main points. First, the relative performance of the off-line + on-line versions and the literature + on-line ones in terms of average ranking is the same observed in short runs: the average ranking here is 23.14 and 29.86, respectively. Second, the relative performance of the literature version, which is quite poor in short runs, becomes comparable to the best on-line versions in long runs. In fact, the parameter settings suggested in the literature have been proposed for obtaining good performance with relatively high computational time available. The best performing on-line version in this case is OFF+SAc.

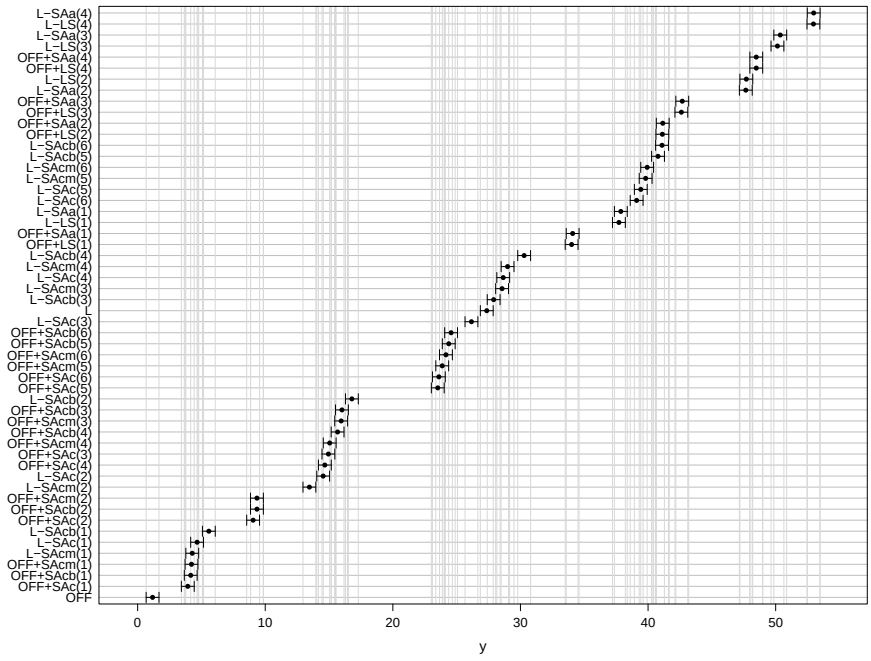
The quality of the results achieved by on-line tuning methods, independently whether they start from the literature or from the off-line tuned parameter setting, worsens as the number of parameters tuned increases. Exemplary results are shown in Figure 2 for SAc on two sets of instances. Here we can observe that, as more parameters are tuned, the ranks for on-line method gets worse, indicating that, the more parameters are tuned on-line, the worse gets the solution quality reached by the algorithms.

5.1.2 Experiment 2: perfect *a posteriori* knowledge on the best parameter(s) to be tuned.

In the context that simulates the situation in which the implementer has perfect *a posteriori* knowledge on the impact of the different parameters on the performance of on-line tuning methods,

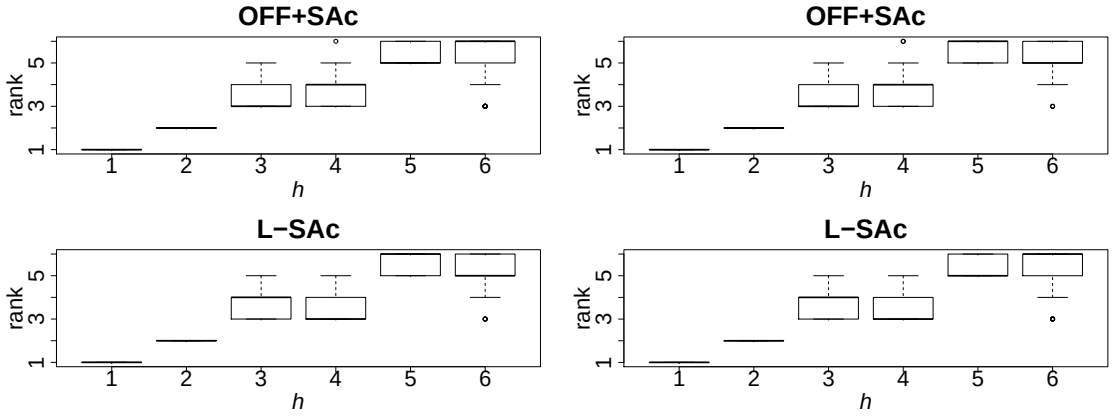


(a) $TSP(2000, u)$



(b) $TSP(x, x)$

Figure 1: **Results simulating no *a priori* knowledge on the best parameter(s) to be tuned.** Simultaneous confidence intervals for all-pairwise comparisons of ranks between all versions of $\mathcal{MM}$AS, applied to two sets of TSP instances. Results for short runs are reported.



(a) $TSP(2000, u)$ (b) $TSP(x, x)$

Figure 2: **Results simulating no *a priori* knowledge on the best parameter(s) to be tuned.** Ranks achieved by an on-line method as a function of the number of parameters tuned. Short runs.

$\mathcal{MM}$AS achieves the results represented in Figure 3. The off-line version performs significantly better than all on-line ones in set $TSP(x, x)$, while this is not true for set $TSP(2000, u)$. The difference is statistically significant only in the latter case. Increasing the level of heterogeneity of the set of instances, then, does not favor on-line methods. As in the context of no *a priori* knowledge on the best combination, the best on-line methods are self-adaptive approaches with parameters managed at the colony level: the best performing is either L-SAcb(1) or L-SAcm(1). Even if in both sets of instances depicted the best method is always a literature + on-line version, in general using the configuration selected by F-Race allows to get to better performance. In fact, the average ranking achieved by off-line + on-line versions is 22.80 in $TSP(2000, u)$, and 22.94 in $TSP(x, x)$. The respective values for the literature + on-line versions are 30.20 and 30.05.

In the sets of instances not shown here, off-line is the best performing method in three out of four sets. The best performing on-line method in all those cases is OFF+SAcm. In long runs on instances of set $TSP(2000, u)$ the best method is L-SAc(1).

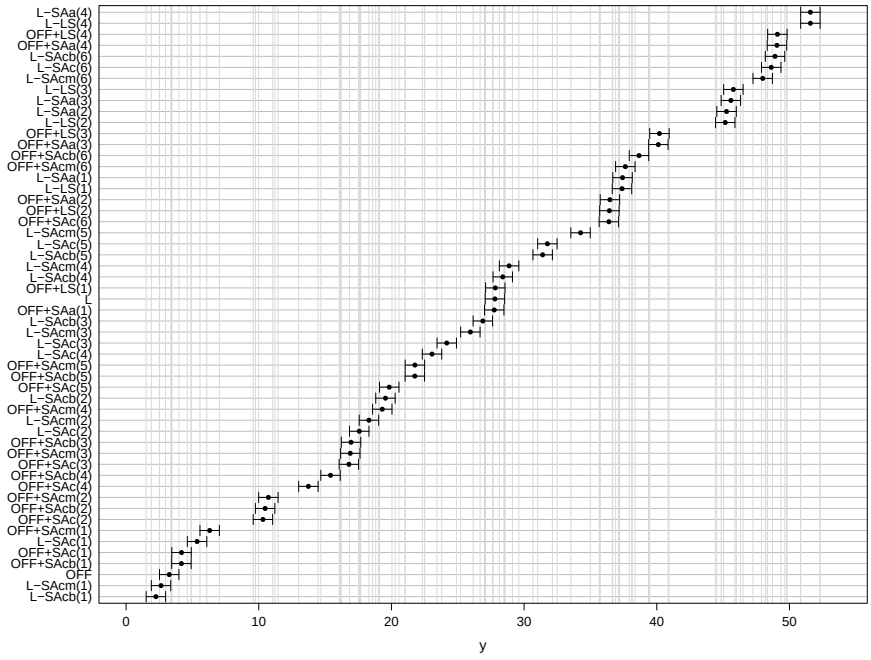
The best on-line methods always tune only one parameter. The quality of the results is clearly decreasing as a function of the number of parameters tuned, as in the case of no *a priori* knowledge on the best parameter(s) to be tuned.

5.2 Quadratic Assignment Problem

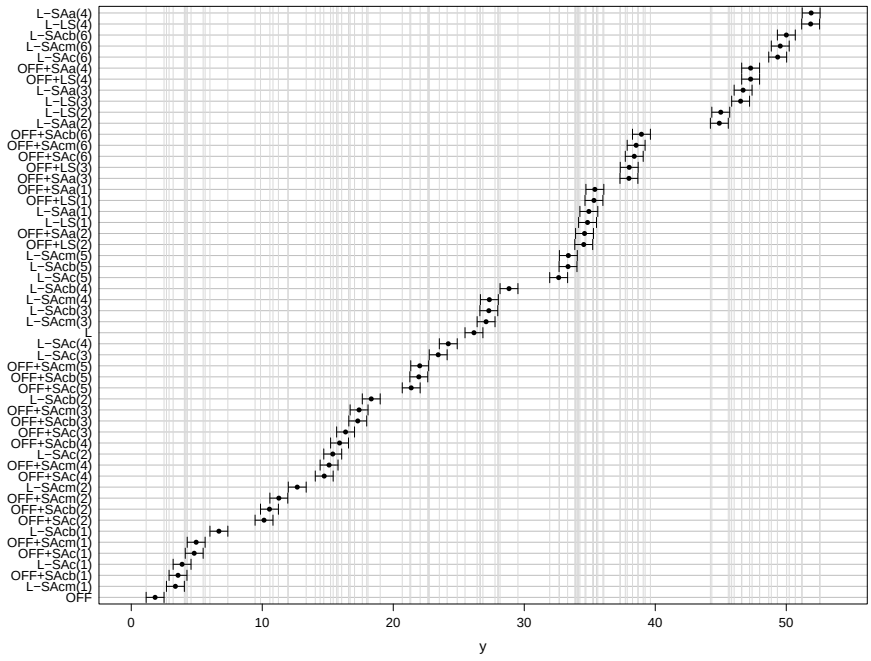
For the QAP, we tackle 28 sets of instances with different levels of heterogeneity. As for the TSP, we report the results of the off-line method and by the five on-line methods on one homogeneous, $QAP(60, ES100)$, and one heterogeneous, $QAP(80, xx)$, set of instances. They are shown in Figures 5 and 7 for the two contexts concerning no *a priori* knowledge and perfect *a posteriori* knowledge on the best parameter(s) to tune on-line, respectively. Figures 6 and 8 describe the ranking of the results as a function of the number of parameters tuned.

5.2.1 Experiment 1: no *a priori* knowledge on the best parameter(s) to be tuned.

The results achieved by all the versions of $\mathcal{MM}$AS implemented in the context of no *a priori* knowledge on the best parameter(s) to tune are reported in Figure 5. In short runs, represented in Figures 5(a) and 5(b), off-line outperforms all on-line versions. The literature version performs very well, in $QAP(80, xx)$ following immediately the off-line version. The difference between off-line + on-line and literature + on-line is negligible when looking at the average ranks, which

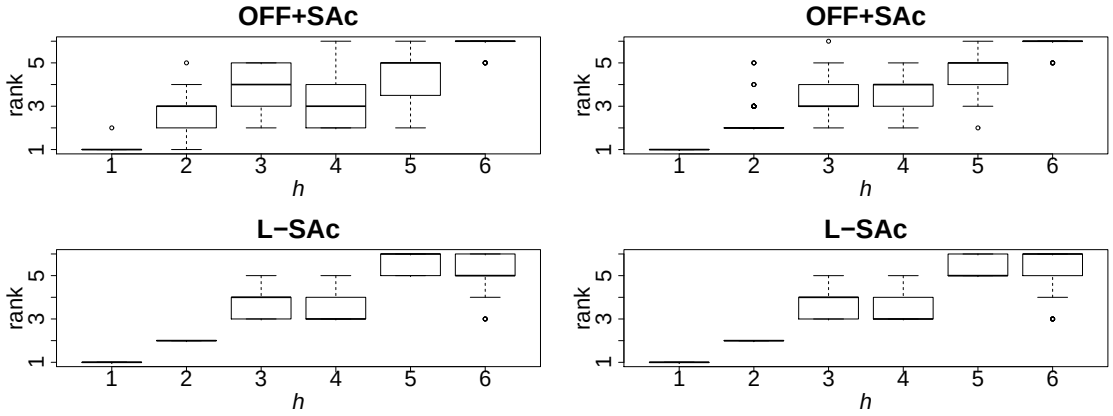


(a) $TSP(2000, u)$



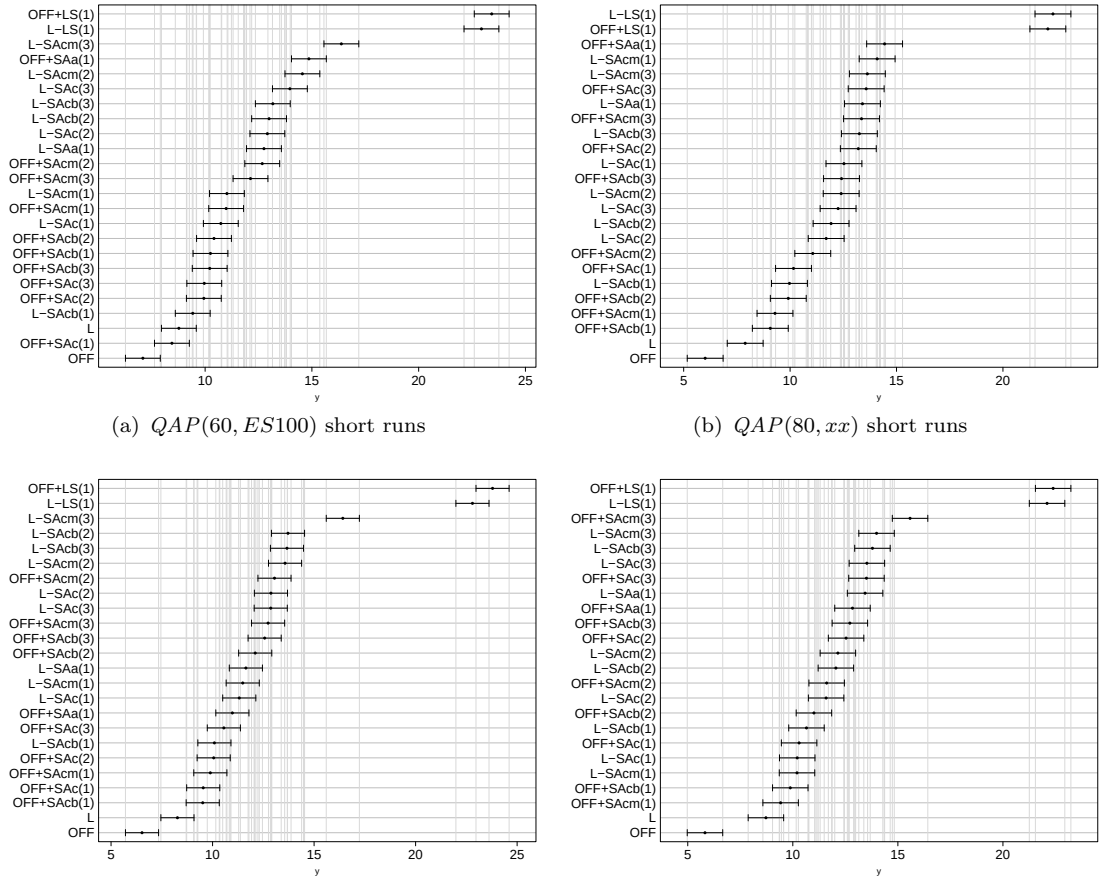
(b) $TSP(x, x)$

Figure 3: **Results simulating perfect *a posteriori* knowledge on the best parameter(s) to be tuned.** Simultaneous confidence intervals for all-pairwise comparisons of ranks between all versions of $\mathcal{MM}$AS, applied to two sets of TSP instances. Results for short runs are reported.



(a) $TSP(2000, u)$ (b) $TSP(x, x)$

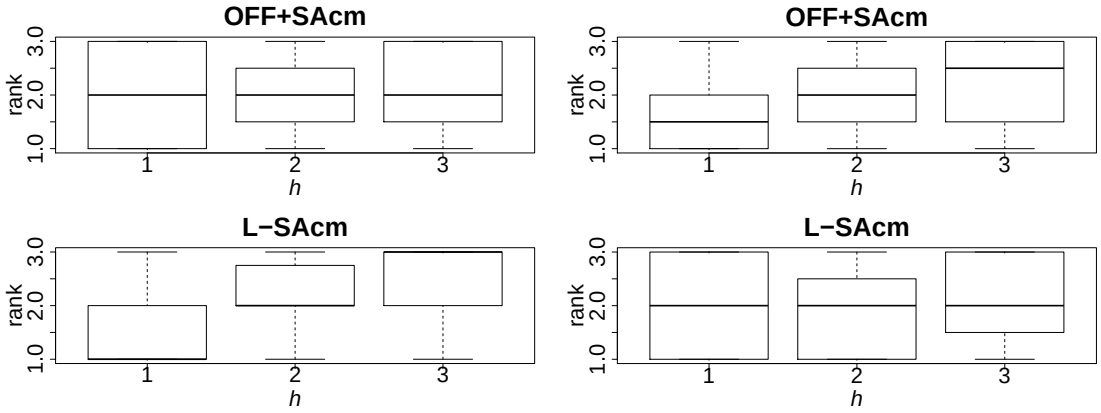
Figure 4: **Results simulating perfect *a posteriori* knowledge on the best parameter(s) to be tuned.** Ranks achieved by an on-line method as a function of the number of parameters tuned. Short runs.



(a) $QAP(60, ES100)$ short runs (b) $QAP(80, xx)$ short runs

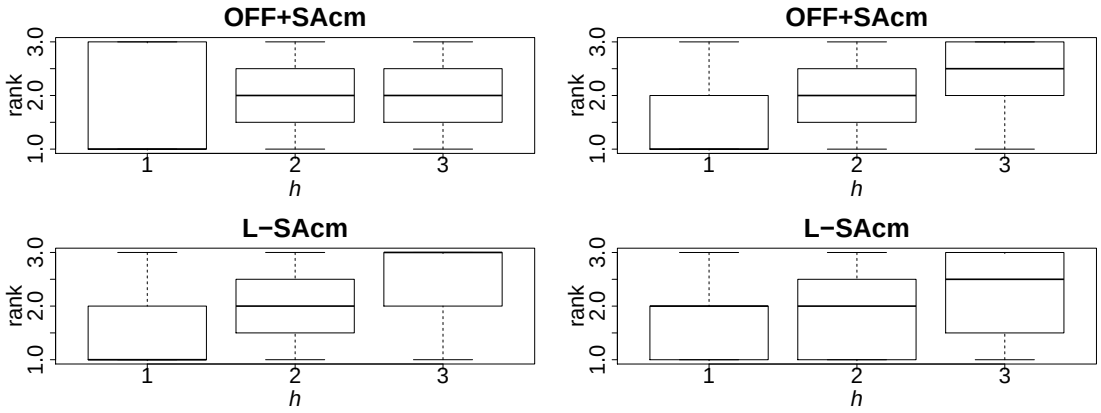
(c) $QAP(60, ES100)$ long runs (d) $QAP(80, xx)$ long runs

Figure 5: **Results simulating no *a priori* knowledge on the best parameter(s) to be tuned.** Simultaneous confidence intervals for all-pairwise comparisons of ranks between all versions of $\mathcal{MM}$AS, applied to two sets of QAP instances.



(a) $QAP(60, ES100)$ short runs

(b) $QAP(80, xx)$ short runs



(c) $QAP(60, ES100)$ long runs

(d) $QAP(80, xx)$ long runs

Figure 6: **Results simulating no *a priori* knowledge on the best parameter(s) to be tuned.** Ranks achieved by an on-line method as a function of the number of parameters tuned.

are 10.79 and 11.14 in sets $QAP(60, ES100)$ and $QAP(80, xx)$ for off-line + on-line. and 12.21 and 11.86 for literature + on-line, respectively. The best on-line method is in both cases a self-adaptive approach with parameters managed at the colony level. This type of on-line methods in general outperforms the method in which parameters are managed at the ant level, LS and SAa. On set $QAP(60, ES100)$, the average rank of the method that manage parameters at the colony level is 10.34; the one of the methods that manage parameters at the ant level is 16.74. The corresponding values on set $QAP(80, xx)$ are 10.44 against 16.28, respectively. In long runs the results are qualitatively very similar, as Figures 5(c) and 5(d) show. The heterogeneity of the sets of instances does not have a remarkable impact on the qualitative results.

If we consider the results on all the sets of instances represented in Pellegrini et al (2010a), we can observe that in 20 out of 28 sets of instances the difference is statistically significant in favor of the off-line version. In 13 sets of instances, even the literature version outperforms all on-line, literature being slightly better than off-line in 5 sets. For what concerns only on-line versions, in nine sets of instances OFF+SAcm achieves the best results, in seven sets the best performing is L-SAcb. OFF+SAcb and OFF+SAc are the best in four sets of instances each, while L-SAc and L-SAcm in one and two sets, respectively.

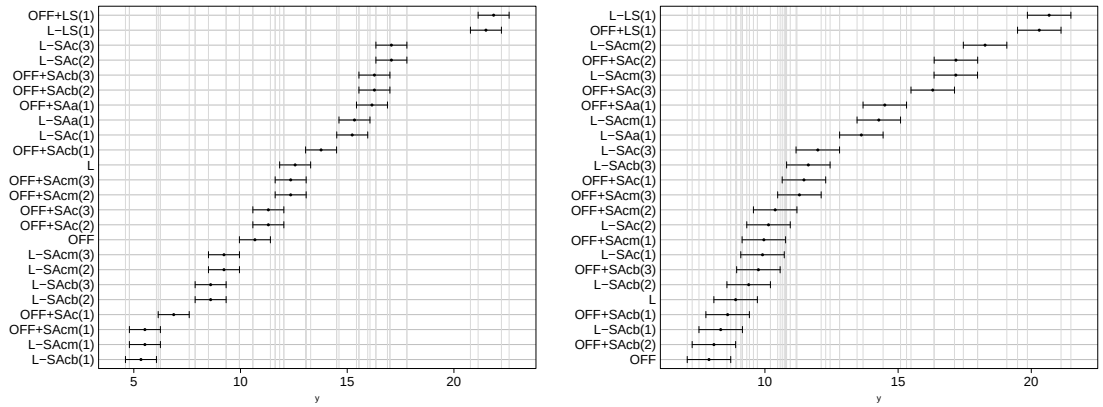
The best performing on-line versions always tune one parameter, and the performance decrease as a function of the number of parameters tuned as shown in Figure 6. The trends are not as clear as for the TSP, possibly owing to the lower number of parameters tuned here.

5.2.2 Experiment 2: perfect *a posteriori* knowledge on the best parameter(s) to be tuned.

For what concerns the context of perfect *a posteriori* knowledge on the best parameter(s) to be tuned, we report exemplary results in Figure 7. In short runs, shown in Figures 7(a) and 7(b), the best on-line versions often outperform the off-line one. In long runs shown in Figures 7(c) and 7(d), the situation is more favorable to the off-line version. As in all other results, the best performing on-line methods are always self-adaptive approaches with parameters managed at the colony level. The average ranking of these methods is 10.29 in $QAP(60, ES100)$, and 10.62 in $QAP(80, xx)$. The corresponding values for the methods that manage parameters at the colony level are 16.97 and 15.48, respectively. In general the off-line + on-line versions perform slightly better than the literature + on-line ones, but the difference in the average ranking is even smaller than in the case of no *a priori* knowledge on the best parameter(s) to be tuned. The best on-line version tunes one and two parameters in set $QAP(60, ES100)$ and $QAP(80, xx)$, respectively. Also in this context, the heterogeneity of the sets of instances does not have a remarkable impact on the results.

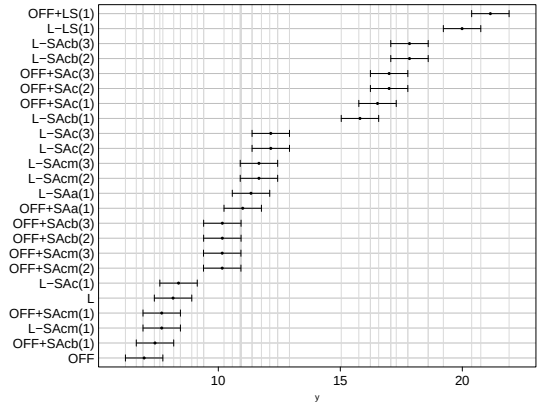
On the instances not represented here, off-line version is seldom the best one in short runs. The difference is significant in favor of either an on-line version in 20 sets of instances, in favor of the off-line version in 2 sets, namely $QAP(80, RR)$ and $QAP(100, RR)$, while in 4 sets the results are comparable. It is not possible to identify one best performing on-line method: L-SAcb obtains the best results in 10 sets of instances, L-SAcm in 7, L-SAc in 4, and OFF+SAcb in 2 and OFF+SAcm in 3 sets. In long runs, in 23 sets of instances the difference between the off-line and the best on-line versions is not statistically significant; off-line is slightly better in 11 sets and slightly worse in 12 sets. The difference is significant in favor of the off-line version in two sets of instances, and in favor of the best off-line + on-line in one set. Considering only on-line versions, OFF+SAcb is the best performing in ten sets of instances, OFF+SAc in five sets, OFF+SAcm in four, L-SAcm and L-SAc in three sets, L-SAcb in two, and OFF+SAa in one set. As it can be observed in the results reported in Figure 7, differently from the context of no *a priori* knowledge on the best subset and from the results on the TSP, using the configuration selected by F-Race as the initial one for on-line tuning is not often convenient here. Moreover, the best on-line version tunes in short runs two parameters in six sets of instances, and even three parameters in set $QAP(100, ES20)$. In long runs, the best on-line method tunes one parameter in all but two sets, in which it tunes two.

As it can be seen in Figure 8, this does not imply an inversion in the trend of the quality of

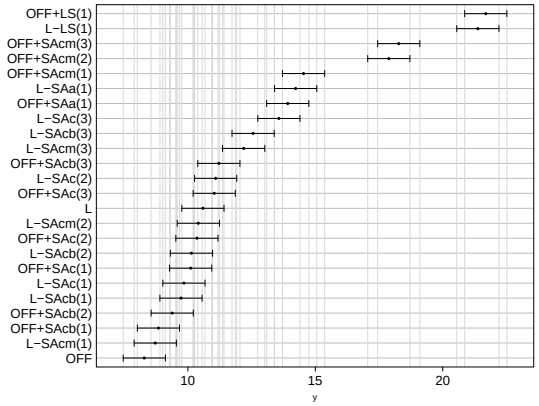


(a) $QAP(60, ES100)$ short runs

(b) $QAP(80, xx)$ short runs

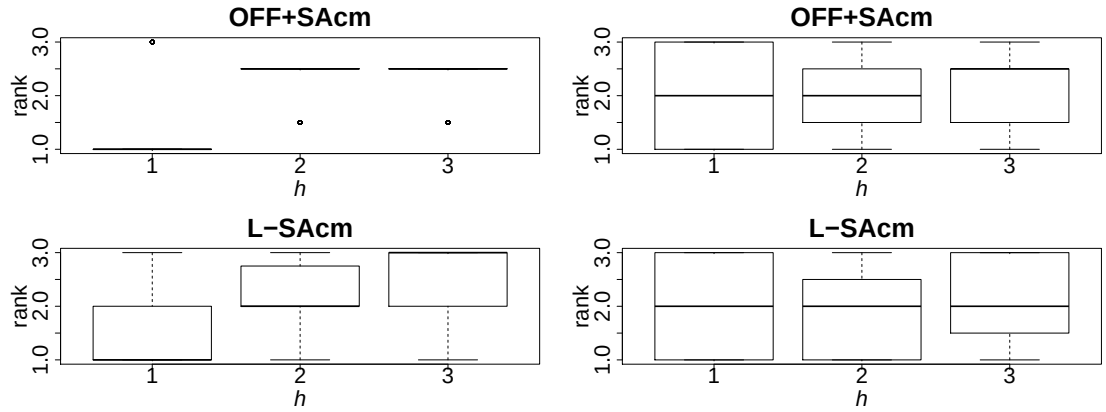


(c) $QAP(60, ES100)$ long runs



(d) $QAP(80, xx)$ long runs

Figure 7: **Results simulating perfect *a posteriori* knowledge on the best parameter(s) to be tuned.** Simultaneous confidence intervals for all-pairwise comparisons of ranks between all versions of $\mathcal{MM}$AS, applied to two sets of QAP instances.



(a) $QAP(60, ES100)$ short runs

(b) $QAP(80, xx)$ short runs

(c) $QAP(60, ES100)$ long runs

(d) $QAP(80, xx)$ long runs

Figure 8: **Results simulating perfect *a posteriori* knowledge on the best parameter(s) to be tuned.** Ranks achieved by an on-line method as a function of the number of parameters tuned.

the results as a function of the number of parameters tuned: even if in some cases this trend is no very evident, tuning few parameters is more convenient than tuning many.

6 Conclusions

In this paper, we study the performance of $\mathcal{MM}$AS when its parameters are tuned off-line and when they are tuned on-line. We run an extensive experimental analysis using the TSP and the QAP as test problems for assessing the relative results achievable with the two tuning approaches.

For on-line tuning we implement five different methods. We observe the results of these methods as a function of the number of parameters tuned, and we consider two different contexts: on the one hand, we simulate no *a priori* knowledge on the best combination of parameters to be tuned on-line; on the other hand, we simulate perfect *a posteriori* knowledge. The second context introduces a strong bias in favor of on-line tuning.

The results presented corroborate two of the three conjectures that typically drive the choice between off-line and on-line tuning: off-line tuning achieves good performance when the instances to be tackled are homogeneous; and tuning on-line few parameters is more convenient than tuning many. Instead, in the context considered here, the results refuse the conjecture according to which on-line tuning is to be preferred when the instances are heterogeneous: the heterogeneity of the instances solved does not have a remarkable impact on the relative results of off-line and on-line tuning.

Moreover, the results show that the configuration used as the initial one in on-line tuning may have a significant impact on the performance. Often, using the configuration selected by off-line tuning appears quite convenient with respect to using the configuration proposed in the literature. This observation, together with the result that on-line tuning can achieve better results when it manages few parameters, suggests that understanding the impact of each parameter on the behavior of the algorithms may help in identifying the most important parameters to be tuned on-line. Achieving the high level of understanding of an algorithm necessary to this aim may require a high effort. This effort may be significantly reduced by exploiting off-line tuning: a tuning approach based on the hybridization of off-line and on-line tuning might boost the performance of optimization algorithms such as swarm intelligence techniques. In particular, the information on the importance of parameters gained in off-line tuning might be used for choosing which parameters to tune on-line. An on-line method might then tune the configuration to adapt to each specific instance and possibly to each phase of the search.

Acknowledgements

This work was supported by the META-X project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium and by the European Union through the ERC Advance Grant "E-SWARM: Engineering Swarm Intelligence Systems" (contract 246939). Mauro Birattari and Thomas Stützle acknowledge support from the Belgian F.R.S.-FNRS, of which they are Research Associates. The authors thank the colleagues that answered the survey described in the paper.

References

Adenso-Díaz B, Laguna M (2006) Fine-tuning of algorithms using fractional experimental designs and local search. Operations Research 54(1):99–114

Anghinolfi D, Boccalatte A, Paolucci M, Vecchiola C (2008) Performance evaluation of an adaptive ant colony optimization applied to single machine scheduling. In: Li X, et al (eds) SEAL, Springer, Heidelberg, Germany, LNCS, vol 5361, pp 411–420

Balaprakash P, Birattari M, Stützle T (2007) Improvement strategies for the F-race algorithm: Sampling design and iterative refinement. In: Bartz-Beielstein T, et al (eds) HM 2007, Springer, Heidelberg, Germany, LNCS, vol 4771, pp 108–122

Battiti R, Brunato M, Mascia F (2008) Reactive Search and Intelligent Optimization, Operations Research/Computer Science Interfaces, vol 45. Springer, Berlin, Germany

Birattari M (2003) Race. R package, http://cran.r-project.org

Birattari M (2004) On the estimation of the expected performance of a metaheuristic on a class of instances. How many instances, how many runs? Tech. Rep. TR/IRIDIA/2004-01, IRIDIA, Universit Libre de Bruxelles, Brussels, Belgium

Birattari M (2009) Tuning Metaheuristics: A Machine Learning Perspective, Studies of Computational Intelligence, vol 197. Springer, Berlin, Germany

Birattari M, Dorigo M (2007) How to assess and report the performance of a stochastic algorithm on a benchmark problem: Mean or best result on a number of runs? Optimization Letters 1(3):309–311

Birattari M, Stützle T, Paquete L, Varrentrapp K (2002) A racing algorithm for configuring metaheuristics. In: Langdon W, et al (eds) GECCO 2002, Morgan Kaufmann Publishers, San Francisco, CA, pp 11–18

Chiarandini M (2005) Stochastic local search methods for highly constrained combinatorial optimisation problems. PhD thesis, Computer Science Department, Darmstadt University of Technology, Darmstadt, Germany, chapter 3.

Clerc M (2006) Ant Colony Optimization. ISTE Publishing Company

Dorigo M, Gambardella LM (1997) Ant Colony System: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation 1(1):53–66

Dorigo M, Stützle T (2004) Ant Colony Optimization. MIT Press, Cambridge, MA

Eiben AE, Michalewicz Z, Schoenauer M, Smith JE (2007) Parameter control in evolutionary algorithms. In: Lobo et al (2007), pp 19–46

Förster M, Bickel B, Hardung B, Kókai G (2007) Self-adaptive ant colony optimisation applied to function allocation in vehicle networks. In: GECCO'07, ACM Press, New York, NY, pp 1991–1998

Hutter F, Hoos HH, Leyton-Brown K, Stützle T (2009) ParamILS: An automatic algorithm configuration framework. Journal of Artificial Intelligence Research 36:267–306

Johnson D, McGeoch L, Rego C, Glover F (2001) 8th DIMACS implementation challenge. `http://www.research.att.com/$\sim$dsj/chtsp/`

Khichane M, Albert P, Solnon C (2009) A reactive framework for ant colony optimization. In: Stützle T (ed) Learning and Intelligent OptimizatioN (LION), Springer, Heidelberg, Germany, LNCS, vol 5851, pp 119–133

Lobo F, Lima CF, Michalewicz Z (2007) Parameter Setting in Evolutionary Algorithms. Springer, Berlin, Germany

Martens D, Backer MD, Haesen R, Vanthienen J, Snoeck M, Baesens B (2007) Classification with ant colony optimization. IEEE Transactions on Evolutionary Computation 11(5):651–665

Pellegrini P, Stützle T, Birattari M (2010a) Companion of tuning $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System with off-line and on-line methods. `http://iridia.ulb.ac.be/supp/IridiaSupp2010-013/`, iRIDIA Supplementary page

Pellegrini P, Stützle T, Birattari M (2010b) Off-line and on-line tuning: a study on $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System for TSP. In: Dorigo M, et al (eds) ANTS 2010: Seventh International Conference on Swarm Intelligence, Springer, Heidelberg, Germany, LNCS, vol 6234, pp 239–250

Randall M (2004) Near Parameter Free Ant Colony Optimisation. In: Dorigo M, et al (eds) ANTS 2004, Springer, Heidelberg, Germany, LNCS, vol 3172, pp 374–381

Stützle T (2002) ACOTSP: A software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem. `http://www.aco-metaheuristic.org/aco-code`, URL `http://www.aco-metaheuristic.org/aco-code`

Stützle T, Fernandes S (2004) New benchmark instances for the QAP and the experimental analysis of algorithms. In: Gottlieb J, Raidl G (eds) EvoCOP 2004, Springer, Heidelberg, Germany, LNCS, vol 3004, pp 199–209

Stützle T, Hoos HH (2000) MAX–MIN ant system. Future Generation Computer Systems 16(8):889–914

Taillard E (1991) Robust taboo search for the quadratic assignment problem. Parallel Computing 17 17:443–455